January 23

2009

introduction to C ++

Explain the concepts in C ++ programming language

Computer structure

Computer Operation

introduction to C ++

1 – Program Structure

2 – Variable And Data Type

3 – Input / Output Statement

4 – Flow Control

5 – Iterative Statement

1 – Program Structure

الهيكل التنظيمى للبرنامج داخل الجهاز او داخل لغة **C** هو كيف يتم كتابة البرنامج بشكل صحيح

فى البداية يجب ان نعرف ان اى برنامج داخل اللغة يحتوى على جزئين اساسيين : -

الاول يسمى **Header** واى برنامج يحتوى على **Header** واحد فقط

الثانى يسمى **Block** واى برنامج يمكن ان يحتوى على اكثر من **Block** على الاقل يجب ان يحتوى على واحد

و **Block** هو عبارة عن دالة **Function** وابسط برنامج يحتوى على **Header** و **Block**

العناصر الاساسية التى تكتب فى **Header** والعناصر التى تكتب فى **Block**

اولا العناصر التى تكتب فى **Header** : -

١ – Directives : -

عبارة عن مجموعة ملفات موجودة داخل مترجم اللغة يتم استدعائها لتصبح جزء من البرنامج وهى مكتوبة مسبقا داخل مترجم اللغة او ما يسمى ب Compiler وهذا الجزء هو جزء محورى واساسى داخل اى لغة برمجة اذ انه عن طرقه يتم التعامل مع اللغة بشكل سهل وبمعرفة كل ما يحتويه هذا الجزء من كلمات محجوزة داخل اللغة وكيفية التعامل معها وفيما تستخدم كل كلمة فاننا نسير بذلك نحو فهم متعمق للغة البرمجة بشكل عام

وجدير بالذكر ان هذا الجزء هو النواة الاساسية الذى بنيت عليه جميع لغات البرمجة كما انه يستخدم وسيط بين اللغة التى يفهمها الشخص واعنى هنا الكود الذى يكتبه المبرمج وليست اللغة العادية وبين اللغى التى يفهمها الجهاز وهى ٠ و١

اذا هناك ثلاث لغات يتم التمحور عليهم مع اى لغة برمجة وهما اللغة التى يفهمها الشخص وهى الكود الذى يتم كتابته بواسط المبرمج وبين اللغة التى يفهمها الجهاز وهى ٠ و ١ والتى يلعب Compiler دور الوسيط الرئيسى فيها

اللغة الثالثة هى التى يفهمها المستخدم العادى وهى لغة الارقام والحروف والكلمات العادية والتى

يتعامل بها مع اى برنامج من خلال نوافذ التحكم لديه والتى ينسق المبرمج الكود الذى يكتبه لخدمة هذه البيانات ولخدمة المستخدم العادى ويلعب هنا المبرمج دور الوسيط فيها

ولذلك نجد انه لكل لغة Compiler خاص بها وهو مكان كتابة الكود داخل اللغة اذ لا يصح استعمال كود ال C مثلا داخل Compiler مخصص لل Basic كما يوجد لاى لغة الكلمات الخاصة بها وطرق خاصة لكتابة الجمل البرمجية وطريقة ترتيبها داخل ال كود

والان نرى الكلمات المتعلقة بلغة++ C : -

include < I / o stream.h >

Include# هى دالة موجودة داخل اللغة ومعناها استدعاء الامر ليصبح داخل Block

Stream اسم الملف الذى تخزن به حيث يتم تخزين الملفات باسماء مختلفة وهذا احد اسمائها

H. هذا هو امتداد الملف وسمى ذلك لانهذا الملف Headmen

I/O تستخدم فى توصيف IN C و OP C لكى يعملوا فى البرنامج بشكل صحيح واذا لم تكتب

داخل **Header** فلن يمكن استخدام الخرج والدخل بشكل صحيح داخل **Block** حيث لا يخلو اى برنامج منهم

على سبيل المثال هناك دالة تسمى <**math**> هذه الدالة عبارة عن ملف داخل اللغة ويحتوى على توصيف الدوال الرياضية بجميع انواعها ولذلك يجب استدعائها فى ال **Header** عند التعامل مع اى دالة رياضية داخل **Block** ولكن يجب ان نعرف ان هناك فرق بين الدالة الرياضية والعملية الحسابية ونشرح ذلك بالمثال التالى : -

A + B; X

هذا الكود عندما يكتب فى **Block** لا يلزم تعريف دالة **math** فى بداية **Header** وذلك لانه عملية حسابية عادية

SIN (X)

عندما تكتب هذه الدالة فى **Block** فانها تلزم تعريف دالة **math** داخل **Header** وذلك لانها دالة رياضية

كيف يتم تحديد الملفات التى تكتب فى **directives** تحدد على اساس الخطوات التى تكتب فى **Block**

كالاتى : -

تحدد على اساس الخطوات التى تكتب فى **Block** كما راينا فى المثالين السابقين اذا فلا داعى للحيرة فى اختيار الكلمات التى ستكتب فى **Header** وذلك لانها معروفة مسبقا وهذه العملية تسمى **Variable Deceleration Global** وهى تعريف المتغيرات العامة المستخدمة داخل **Block** او داخل اكثر من **Block** وهذا هو الجزء الثانى من عمل **Header** داخل اللغة وهذه المتغيرات التى تستخدم على مستوى البرنامج ككل بمعنى ان لها صلاحية الاستخدام داخل اى جزء من اجزاء البرنامج .

الان ننتقل الى وصف **Block** واجزائه : -

اى **Block** داخل البرنامج هو دالة ولابد ان يكون مميز باسم واذا كان البرنامج يحتوى على

Block واحد لابد ان يكون اسمه {}**main** او اسم الدالة {}**main** ولو كان هناك اكثر من

Block اجبارى ان يكون اسم واحد منهم {}**main** لان مرحلة التنفيذ ستبداء من خلاله

والان نرى الهيكل العام لاى **Block** داخل اللغة : -

```
main { }
{
    Deceleration Local Variable
        Statements ;
        return ( )
}
```

وسنشرح الان كل جزئية من هذا ال**Block**

- **main { }** : - هو اسم الدالة او **Block**

- **{** هذا القوس يعبر عن البداية او **Begin** ويعلم هنا ال**Compiler** بان عملية كتابة

ال**Block** ستبداء

- **Variable Deceleration Local** هى تعريف النتغيرات المحلية التى ستتخدم
- داخل **Block** وهى عبارة عن مجموعة جمل او **Statements** ويجب بعد الانتهاء

من كتابة اى جملة داخل **Block** وضع ؛

- **return ( )** تستخدم كاشارة الى مترجم اللغة ان كل الخطوات السابقة قد تم كتابتها

- بشكل صحيح

- } هذا هو قوس النهاية او **End** ويعلم ال **Compiler** بان هذا ال**Block** قد تم
- الانتهاء منه

بالنسبة الى جملة ( ) **return** اذا كان القوس الخاص بها لا يوجد به شيء اذا لابد ان نكتب قبل اسم الدالة { } **main** كلمة **Void** لتصبح { } **void main**

ولو كان يوجد به (٠) يجب ان نكتب **int** لكى تصبح { } **int main**

ولاحظ الكتابة داخل القوس هى شيء اختيارى ولا يمت باى صلة الى صحة الكود او خطائه

ولعلنا لاحظنا الفرق هنا بين **Global** و **Local** فى تعريف النتغيرات

Global هى تعريف للمتغيرات العامة على البرنامج وممكن ان تستخدم فى جميع **Blocks**

Local هى متغيرات محلية اى يمكن استخدامها داخل **Blocks** التى تعرف فيه ولا تخرج عن هذا الحد

#والان ننتقل الى شرح انواع المتغيرات وانواع البيانات والتى تستخدم فى كل لغة على حسب طرق تعريفها

ما هو المتغير : - اى متغير هو كلمة مكونة من مجموعة حروف ومجموعة ارقام ونرى ذلك فى

المثال التالى

Xx متغير من حروف

X1 متغير من حروف وارقام

المهم هو ليس اسم المتغير اكتب ما تشاء ولكن المهم هو ان يعرف بشكل صحيح ويختار له نوع من انواع البيانات المعرفة داخل اللغة ويتم الالتزام بكل ما يحتويه هذا النوع من البيانات من شروط وضع القيم له وطريقة التعامل معه وكيفية استخدامه داخل اللغة .

كيف نعرف المتغير **Deceleration Variable** : -

يوجد قاعدة عامة داخل اللغة لتعريف المتغيرات وهى

data type Var name ;

نوع البيانات (مسافة) اسم المتغير

ونرى الان بعض الامثلة لتعريف متغيرات ووضع ملاحظات لفهم ماهية تعريف هذه المتغيرات داخل اللغة

Int x;

تعريف متغير اسمه **x** ويشير الى نوع بيانات **int** وهذه الخطوة عندما تنفذ تخبر مترجم اللغة او **Compiler** ان يبحث عن اقرب مكان فارغ فى الذاكرة ويحجزه ويشير اليه بالمتغير **x** حتى تستخدم لادخال البيانات ثم قراءتها لاحقا لعمل عليها عمليات مختلفة ونوع بيانات **int** هو نوع لتعريف الارقام الصحيحة الموجبة القصيرة

Float y;

ومعناها تعريف متغير اسمه **y** ويشير الى بيانات من نوع **float** اى بيانات ارقام تحتوى على كسور موجبة وان يحجز له مكان فى الذاكرة ويشير اليه بالمتغير **y**

احجام الاماكن داخل الذاكرة : - الحجم يعتمد على نوع المتغير فمثلا

اذا كان **int** ياخذ ٢ **byte**

اذا كان **float** ياخذ ٤ **byte**

اذا كان **char** ياخذ ١ **byte**

....

ولكن ما هى انواع البيانات ككل وكيف نتعامل معها داخل اللغة : -

البيانات بصفة عامة داخل لغة البرمجة تنقسم الى قسمين رئيسين وهما : -

١ – **built in statements**

٢ – **user define statements**

١ – هى انواع بيانات معرفة داخل اللغة مثل **int ، char , float**

معنى كلمة معرفة داخل اللغة او كما يطلق عليها **word reserved** انها كلمة يعرفها المترجم ولايحتاج من المبرمج ان يعرفها له بل هى اساسا تستخدم لتعريف اسماء اخرى يتكتبها المبرمج باختياره

٢ – هى انواع بيانات غير موجودة فى اللغة ولكن يقوم المبرمج بتعريفها ويعمل لها توصيف داخل اللغة بنفسه مثل **struct** ، **pointer, matrix**

وتعريف اى متغيرات يعتمد على **statements** ولذلك تعليم البرمجة هى عملية معرفة كتابة

Statements داخل اللغة لان كل شىء يعتمد عليها فمثلا نجد ان جملة **if** هى جملة تؤدى نفس الوظيفة داخل كل لغات البرمجة ولكنها تختلف فى طريقة كتابتها والتعامل معها من لغة الى اخرى وهذا هو الفرق الجوهرى بين لغات البرمجة المتعددة

ننتقل الان الى جزئية اخرى وهى المعاملات المستخدمة لبناء عمليات حسابية وتسمى

Operators

Math operators +، - ، *، /، %

جدير بالذكر ان رمز % هو رمز يستخدم لحساب باقى القسمة ويسمى **module** على سبيل المثال

x = ٥/٢ = ٢ **and over 1, so module** = ١

دراسة العاملين الاكثر شيوعا وهما + ، - ويسمى ب **increment** ، **decrement**

++ Increment

-- Decrement

هكذا تكتب داخل مترجم اللغة ++ ، --

الاولى معناها ++ يستخدم لاضافة واحد صحيح على القيمة السابقة

الثانية- يستخدم لحذف واحد صحيح من القيمة السابقة

ولكن متى نضيف ومتى نحذف وسنوضح ذلك فى المثال التالى : -

X = 5;

X ++ = 6 **++ x = 6**

Y = x ++ **y = ++ x**

نلاحظ وبكل بساطة ان استخدام المعامل (++) قبل او بعد المتغير يتم اضافة واحد صحيح على

قيمة المتغير السابقة طالما ان المتغير لم يدخل فى عملية حسابية اما اذا دخل فى عملية حسابية

مثل المعادلتين اسقل

فان الوضع مختلف كالاتى : -

المعادلة الاولى $y = x ++$ هنا قيمة **y** تساوى ٥ وذلك لان معنى هذه الجملة هى مساواة **Y** ب **x** ثم اضافة واحد صحيح على قيمة **x**

المعادلة الثانية $y = ++ x$ هنا قيمة **y** تساوى ٦ لان معنى هذه الجملة هى اضافة واحد صحيح على قيمة **x** ثم مساواة **y** ب **x** اذا يجب التفكير منطقيا اولا ثم حساب القيم فى هذه الحالة

FLOW CONTROL STATEMENTS

وظيفة هذه الجمل انها عبارة عن مجموعة جمل تستخدم فى التحكم فى خطوات تنفيذ البرنامج وترتيب التنفيذ ونجد ان التنفيذ يبداء من **main** ثم يتبع البرنامج خطوة بخطوة بالترتيب ولذلك ممكن تغير هذا الترتيب باستخدام هذا النوع من الجمل وممكن ايضا تنفيذ خطوة دون الاخرى باستخدام شرط معين

وهذه الجمل هى : -

IF STATEMENTS

IF ELSE STATEMENTS

NESTED IF STATEMENTS

SWITCH STATEMENTS

IF STATEMENT

القاعدة العامة لكتابة هذا النوع من الجمل : -

If (condition)

Statement;

Or

{Block of statements}

هذه الصيغة معناها انه لو تحقق الشرط يتم تنفيذ الجملة بشكل صحيح ويجب ان يوضع الشرط بين اقواس

اذا اردنا تحقيق عدة جمل فيجب عمل لها قوس بداية ونهاية كالتالى : -

```
If   (condition)

{
  Statement 1;
  Statement 2;
  Statement 3;
}
```

For example: -

```
If        (A > B)

  Cout << A;

  If   (A > B)

{

  Cout << A;

     A ++;

 Cout   << A;

      }
```

IF ELSE

وتستخدم لاختيار جزء واحد من جزئين فلو تحقق الشرط سيتم تنفيذ الجزء الاول ولو لم يتحقق سيتم تنفيذ الجزء الاخر

القاعدة العامة : -

```
If    (condition)
        Statement;
    Else
        Statement;
```

For example: -

```
If        (A > B)
{
    Cout << A;
    A ++;
}
Else
    Cout << B;
```

اذا لم يتم وضع الاقواس سيعتبر البرنامج خطاء ويتوقف عن التنفيذ وذلك لان البرنامج سيعتبر **A++** هو شرط اخر دون كتابة **IF** ولذلك يجب وضع الاقواس وذلك لان كل ما بداخله ينتمى الى **IF** ويعتبر جملة واحدة

NESTED IF

تستخدم فى حالة تنفيذ خطوة او عدة خطوات فى وجود اكثر من شرط وتسمى جملة شرطية متداخلة او تسمى **if** المتداخلة

القاعدة العامة : -

```
If (condo 1)
    If (condo 2)
        If (condo 3)

            Statement 1;

                Else

            Statement 2;
```

وتطابق هذه الصيغة الصيغة الاتية : -

```
If      (condo 1 & condo 2)
{
        If      (condo 3)
            Statement;
    }
Else
                Statement;
```

الصيغة الاولى : - معناها اذا تحقق الشرط الاول والثانى والثالث فسيتم تنفيذ الجملة واذا لم يتحقق واحد فقط منهم سينتقل الى **else**

الصيغة الثانية : - اذا تحقق الشرط الاول والثانى معا فسيتم اختبار الشرط الثالث لو تحقق فسيتم

تنفيذ الجملة واذا لم يتحقق اى من الشرطين الاول او الثانى فسوف ينتقل الى **else** دون ان يختبر الشرط الثالث ولاحظ ان **else** تنتمى لاقرب **if** كما فى المثال التالى

```
If      (condo 1)

            Statement 1;

        If   (condo 2)

            Statement 2;

Else

            Statement;

Else

            Statement;
```

SWITCH STATEMENT

تستخدم للتحكم فى سير البرنامج وتستخدم لاختيار تنفيذ جزء من عدة اجزاء فمثلا لو عندى ١٠ اجزاء واردنا اختيار جزء واحد اذا نستخدم جملة switch

القاعدة العامة : -

```
Switch   variable

{

Case 1: statement;
        Break;
Case 2: statement;
        Break;
Case 3: statement;
        Break;
Case n: statement;
        Break;
Default: statement;
}
```

لاحظ هنا ان ما بعد **switch** هو متغير واشرط يكتب بعد **CASE** وممكن كتابة جملة واحدة او مجموعة جمل بين اقواس مع **case** وممكن كتابة عدد **n** من **case**

كيفية اختيار الحالة التى تنفذ : - هى الحالة التى تنطبق القيمة فيها مع قيمة المتغير المعرف سابقا

وما بعد **case** يكتب على اساس نوع المتغير كما فى المثالين التالليين : -

```
Switch    (x)
    X = 10;
{
    Case1,
        Break
    Case2,
        Break
    Case10,
        Break

}
```

```
Switch      (y)
    Y = +;
  {

Case +,
     Break
 Case -,
     Break

       }
```

فى المثال الاول نجد ان الحالة التى تنفذ هى **case** ١٠ لانطباقها مع الشرط

فى المثال الثانى نجد ان الحالة التى تنفذ هى **case** +

Break: - تكتب بعد كتابة جملة **case** حتى يخرج البرنامج خارج **loop** بعد الانتهاء من خطوات كل **case** واذا لم تكتب سيختبر الشرط الصحيح ثم ينفذ كل الجمل ولن يتوقف

Default - الحالة الافتراضية التى يتم تنفيذها اذا لم يتحقق اى **case** او اى شرط وفى هذه الحالة يتم تنفيذ جملة **default**

كيفية التفكير فى كتابة برنامج

وسنقوم فى هذا المثال بشرح برنامج يحاكى الالة الحاسبة ونرى كيف يمكن تقسيم البرنامج على مراحل

يبداء التفكير اولا فى ماهية البرنامج وفيما تستخدم الالة الحاسبة البسيطة سنجد انها تقوم باربع عمليات اساسية وهى + ، - ، * ، / الجمع والطرح والضرب والقسمة ونجد ان الالة لاتنفذ الا عملية واحدة فقط على القيم المعطاة فى كل مرة فاذا وجد عملية معقدة تنفذ حسب الترتيب المنطقى للعملية ولاتنفذ الا واحدة فقط ثم تنتقل الى الاخرى وطالما انها ستختار من متعدد اقصد هنا اختيار نوع العملية فالجملة الامثل هنا هى جملة **switch** وسنقوم الان بتوصيف البرنامج او طريقة تحليل البرنامج

I / p	process	o/p
first	+,-,*,/	result

نجد التوصيف هنا يبداء كالتالى : -

اولا الدخل او **I / p** : -وهو يمثل فى هذا البرنامج قيمتين ومعامل وسنعطى تلك الاسماء لهم

X, y, op

ثانيا العملية **process** : -وهى العملية التى تتم على المدخلات وسنقوم فى البرنامج بسرد لجميع العمليات التى يمكن ان تتم وبالتالى سيكون هنا **Var** لجملة **switch** هو **op** وذلك لانه سيتم اختيار نوع العملية بناءا على قيمته التى سيضعها المستخدم والشروط ستكون هى +و-و*و/ والتى ستكون فى جمل **Case** اما الخرج سيكون هو **R**

الخرج **O/P** : -وهو **R** وسيتم اختيار علمية واحدة عن طريق **SWITCH**

وبعد الانتهاء من هذا التوصيف سنبداء فى كتابة الكود لكل جزء على حدى

١ – اولا الخطوة التى تسمح بادخال المتغيرات

```
CIN >> X;
CIN >> Y;
CIN >> OP;
```

٢ – كود اختيار العملية التى تنفذ والتى تحدد طبق ل **op** الذى سيقوم المستخدم بادخاله

```
Switch (op)
{

Case +: R = X+Y;
        Break ;
Case - : R = X – Y;
        Break;
Case *: R = X * Y;
        Break;
Case /: R = X / Y;
        Break ;
Default: Cout << "no operator";

}
```

جملة الخرج والتى ستطبع قيمة الخرج **R** على الشاشة : -

```
Cout << R;
```

ويكون البرنامج فى صورته النهائية كالاتى : -

```
# include < 1 / o stream.h >
Void main {}
{
Int X, Y, R;
Char OP;
CIN >> X;
CIN >> Y;
CIN >> OP;
Switch (op)
{
Case +: R = X+Y;
Break ;
Case - : R = X – Y;
Break;
Case *: R = X * Y;
Break;
Case /: R = X / Y;
Break ;
Default: Cout << "no operator";
}
Cout << R;
Return ();
}
```

ITERATIVE STATEMENT

مجموعة خطوات تستخدم لتكرار جزء داخل البرنامج ويوجد ٣ انواع من هذه الجمل

1 - FOR LOOP

1 - WHILE LOOP

1 - DO WHILE LOOP

وسنرى الان القاعدة العامة لكل منها : -

FOR LOOP

For (initialization, condition, and update);

Statements;

OR

{Block of statements}

For (0, I < 10, I ++);

Cout << I;

While Loop

معناه ان طالما الشرط محقق نفذ العملية

<u>القاعدة العامة</u>: -

While (condition)

Statements;

OR

{Block of statements}

هذاالتركيب صحيح ولكن استخدامه على هذا النحو فى البرنامج خاطىء والسبب فى هذا المثال

While (I < 10)

Cout << I;

افترض فى هذا المثال اننا وضعنا I = ٥ اذا سيظل البرنامج يطبع الرقم ٥ على الشاشة دون ان

يتوقف وذلك لان لاشرط سيظل صحيح ولن ينتهى **loop** ولذلك يجب ان يكون هناك **update** حتى يصل الى قيمة معينة لا تحقق الشرط وبالتالى ينتهى **loop** وتتحول الصيغة الى الاتى

```
While    (condition)
{
  Statements;
    Update;
}
```

انفترض مثلا ان المستخدم قد ادخل قيمة تفوق الشرط مثلا ٢٠ وكان الشرط اقل من ١٠ اذا سيتوقف البرنامج قبل ان يبداء ولذلك يجب ان يكون هناك قيمة ابتدائية قبل **while** اذا لابد ان يتوافر ، **condition, initialization , update** ولابد ان تتوافر فى اى جملة **Iterative**

وتكون الصيغة النهائية كالتالى : -

```
Initialization;
While   (condition)

{
Statements;
Update;
}

I = 0
While   (I < 10)
{
Cout << I;
I ++;
}
```

الفرق بين **for** و **while** ان الاولى تستخدم اذا كان عدد مرات التكرار ثابت والثانية اذا كان

غير معروف ولاحظ انه ممكن عمل **loop** غير منتهى مع وجود شرط وذلك عندما يكون استخدام البرنامج عدد مراته غير محدده مثل برنامج الالة الحاسبة كالاتى

Char s;

While (S ≠ OFF)

{

Perform program

}

ما حدث هنا اننا قمنا بتعريف مفتاح **on** ، **off** على هيئة **S**

ومعنى هذا البرنامج انه طالما المستخدم لم يضغط على ذر اطفاء الالة سيشرع الجهاز بتنفيذ اى اومر يتلقاها فى البرنامج الالة الحاسبة الى ان يغلق

STRUCTURE AND ARRAYS

STRUCTURE

1 – STRUCTURE DECLARATION

2 – ASSIGNMENT WITH STRUCTURE

3 – COMPLEX STRUCTURE

ARRAYS

1 – DECLARATION

2 – ACCESSING ARRAY ELEMENTS

3 – INITIALIZATION

4 – Multidimontial Array

5 – Array Of Character

6 – Array Of Structure

Structure

هيكلة او تنظيم البيانات داخل الكمبيوتر مثلا كتابة بيانات عن الطلاب نجد ان كل صف داخل جدول يمثل بيانات طالب واحد وهذا الصف يمثل **struct** وجميع الصفوف تمثل جدول وهذا الجدول يوصف ب **Array** ومن الممكن اختلاف عدد الاعمدة اى قابلة للزيادة

Struct يناظر صف واحد داخل الجدول ويسمح بادخال وحدة واحدة فقط اى بيانات طالب واحد فقط و هو سجل يكفى لتخزين بيانات وحدة واحدة وهو مقسم الى مجموعة حقول وتحتوى على بيانات مختلفة النوع

نجد ان كل **struct** باسم داخل البرنامج ويميز باسم لكل حقل داخل البرنامج ونرى الان على سبيل المثال

Student

Name	City	Birth	Numbe -

تجد ان لكل حقل اسم خاص به وايضا ان هناك اسم **struct** وهو **student**

Declaration

كيف يتم توصيف **struct** داخل البرنامج

القاعدة العامة

Struct struct name

{

Field 1 deceleration

Field 2 deceleration

Field n deceleration

}

اولا يتم كتابة الكلمة المعرفة داخل اللغة **struct** ثم ترك مسافة ثم كتابة اسم له بعد ذلك فتح قوس للبداية ثم توصيف جميع الحقول داخل **struct** وبعد الانتهاء يتم غلق القوس

ولنا المثال التالى : -

```
Struct          stedata
 {
   Int   no;
   Char      name;
   Char       city;
    Int       birth;

  }
```

كيف يتم استخدام هذا الوصف داخل البرنامج : -

لابد من تعريف متغير من نوع **defined user** وسيعرف لخدمة **stedata** فى المثال السابق

لاحظ ان كلمة **stedata** هى مجرد اسم ل **struct** ولكن مازالت مجهولة ل**compiler**

ويتبقى تعريفها

اولا على انها متغير من نوع **defined user** لكى تكتمل حلقة التعريف

```
Stedata      S1, S2, S3;
```

ونجد هنا ان كلا من **s1,s2,s3**، يمثل **struct** مكون من ٤ حقول وسيتم استخدام المتغير

فقط **S** دون استخدام **stedata** داخل البرنامج

وبذلك قد تم تعريف متغيرات من نوع **defined user** وقد تم تحديد نوعها على انها

Struct

ASSIGNMENT WITH STRUCT

كيف يتم التخزين فى حقول **struct** وقرائة بيانات منها وكيف يتم تنفيذ عمليات على البيانات بداخلها

القاعدة العامة للتعامل مع اى حقل داخل **struct** : -

Variable of Struct Type. Field name

وبالتالى نستطيع ان نشير الى اى حقل معين داخل **struct**

```
CIN >> S1.NO;
NO = 50;
```

نشير الى الحقل **no** داخل **S1** وقمنا بادخال قيمة = ٥٠

```
CIN >> S1.NAME;
```

ونشير الى الحقل **name** داخل **S1** ولاحظ ان الاسم مكون من عدة حروف وقد تستخدم مصفوفة من الحروف فى هذه الحالة

وهناك طرق اخرى لادخال البيانات

```
S1.CITY = CIRO;
```

قراءة البيانات من الحقول

```
COUT << S1.NO;
ZZ = S1.CITY;
```

كيفية اجراء عمليات حسابية على البيانات : -

سنفترض مثلا ان رقم مسلسل خطاء ٥٠ وهو ٥٥ اذا سيتم تعديل الرقم باضافة ٥ وعملية جمع

```
S1.NO = S1.NO + 5;
```

معناها جمع محتوى الحقل no الموجود داخل S1 على ٥ وخزنه مرة اخرى داخل الحقل no

Complex Struct

OR

Struct Of Struct

افترض انك تسجل على البرنامج بيانات الاصدقاء ووجدت مثلا ان حقل الهاتف يحتوى على اكثر من بيان مثل رقم المنزل والعمل والرقم الخاص وبهذه الطرقة لا يمكن وضعهم داخل حقل واحد

وهنا ياتى دور **struct complex**

اذا هو تعريفه هو سجل احد حقوله يحتوى على **struct** وممكن ان تكون مفتوحة

كيف يتم بناء **struct complex** داخل البرنامج : -

وصفه هو تعريف متغير من نوع **struct** وسنبداء من الداخل الى الخارج اى توصيف **Struct** الداخلى

اولا ثم توصيف الخارجى ونرى ذلك فى المثال التالى : -

فى البداية سيتمك تعريف **struct** اسمه **phone** ووصف جميع حقوله

ثانيا سيتم تعريف الخارجى او الاساسى واسمه **friend** ويتم تعريف **telephone** بواسطة **phone Struct** وبذلك نكون قد عرفنا **phone** على انه جزء من **friend** ومعنى ذلك ان **Struct** الاساسى والذى يستخدم فى البرنامج والذى يتم تعريفه على انه متغير من نوع **user** **Difind** هو **friend**

```
Struct      phone
{
Int   fix;
Int   mob;
};

Struct       friend

{
Int    no;
Char    name [10];
Phone  tel;
}

Friend  f1,f2,f3;
```

كيف يتم يخزين بيانات وقرائتها من COMPLEX : -

اولا حقل داخل سجل مباشر : -

```
CIN >> F1.NO;
```

ثانيا حقل داخل حقل داخل سجل : -

```
CIN >> F1.TEL.FIX;

CIN >> F1.TEL.MOB;
```

يتم اضافة (.) بعد كل حقل اذا زاد عدد الحقول الى لا محدود

ملاحظ على المثال التالى : -

اكتب برنامج لخمسة طلاب داخل صف واحد او سجل واحد

اولا من غير الممكن تسجيل بيانات اكثر من طالب داخل سجل واحد اذا سنلجاء الى تكوين **Struct** من خمسة حقول وتعريف ٥ متغيرات وادخال كل طالب على حدى وذلك لان **struct** لا يسمح الا بادخال كل طالب على حدى

ARRAY

المصفوفات : - هى مجموعة عناصر تحتوى على بيانات متشابهة النوع

نوع البيانات لكل عناصر المصفوفة واحد واى وصفوفة تتميز ب **3** عناصر هى

Name index data type;

ومكن ان تكون **type data** هى من نوع **in built** او من نوع **difind user**

عدد عناصر المصفوفة هى **size - 1** وشرط اساسى ان يكون **size** رقم صحيح موجب **int**

DECELERATION ARRAY

القاعدة العامة

Data Type Array Name [size];

Int X[5];

معنى هذه الجملة ان حجز مكان فى الذاكرة والاشارة على بدايته بالعنوان **x** وهذا المكان يكفى لتخزين بيانات من نوع **int** ومكون من خمسة عناصر وهذه العناصر تناظر خمسة متغيرات هما

X 0 ,x 1,x 2,x 3,x 4

ACCESSING ARRAY ELEMENTS

كيف نتعامل مع عناصر المصفوفة من تخزين بيانات وقراءة وعمليات عليها

يتم استخدام اسم **Array** و **index** كالتالى

CIN >> X[2];

يشير الى عنصر بداخل المصفوفة **X** المميز ب **2 INDEX**

قراءة البيانات : -

COUT >> X[3];

العمليات عليها : -

X[4] = X[2] + X[0];

اذا اردنا ادخال قيمة واحدة لكل عناصر المصفوفة هنا لا يتم استخدام جملة الادخال العادية لانها ستكون مكررة ولن تكون منطقية اذا ياتى هنا دور loop for ويتم استبدال الارقام بمتغير وادخاله داخل loop

CIN >> X[I];

```
For      (I=0 , I ≤ 4 , I++);
CIN >> x[I];
```

TH IS BOOK IS THE PART 1

BUT I WELL INSERT THE PART 2 NEARLY

SOFYANY

MEMORYCODE_84 @YAHOO.COM

Filename:	introduction to c
Directory:	C:\Documents and Settings\sofyany\My Documents
Template:	Normal
Title:	introduction to C ++
Subject:	Computer structure
Author:	(MISHO)
Keywords:	
Comments:	
Creation Date:	٢٠٠٩/٠١/٢٣ ١١:٤٣:٠٠ م
Change Number:	28
Last Saved On:	٢٠٠٩/٠١/٢٤ ١١:٤٠:٠٠ م
Last Saved By:	(MISHO)
Total Editing Time:	380 Minutes
Last Printed On:	٢٠٠٩/٠١/٢٤ ١١:٤١:٠٠ م
As of Last Complete Printing	
Number of Pages:	53
Number of Words:	3,419 (approx.)
Number of Characters:	19,494 (approx.)